COMPUTX REPORT 1

-Achintya Harsha IMT2021525

PROJECT 4

- Implementation of multiplication and fill programs using hack HDL
- Code implemented using CPUemulator.sh and tested using the respective .tst file

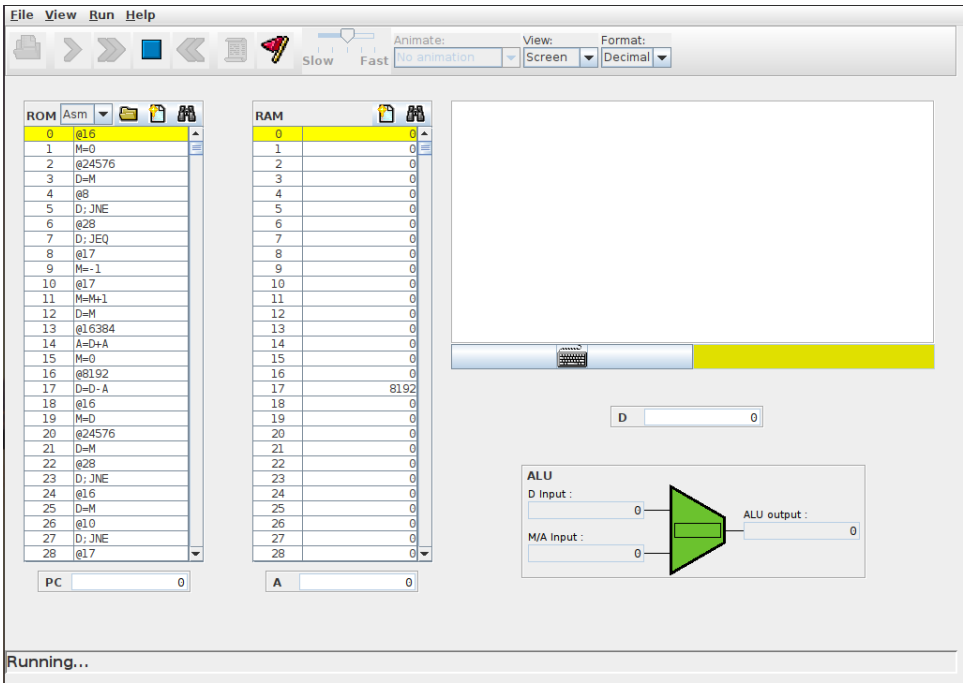
Fill.asm

Algorithm:

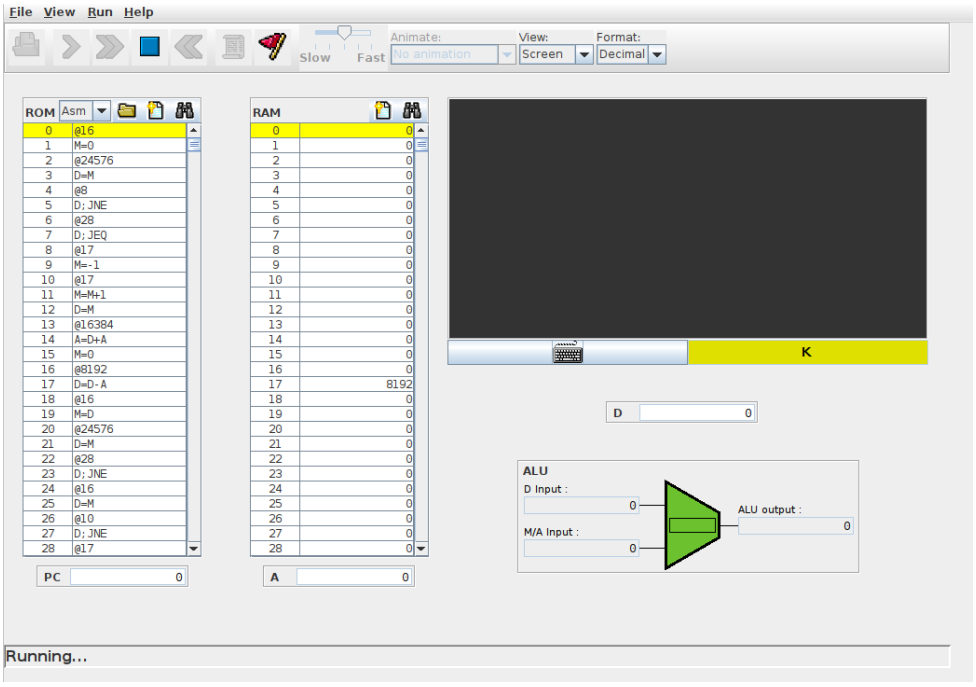
- An infinite loop is created. In this loop keyboard input is taken and if key is pressed, the control is shifted to the FILL label. Else if key is not pressed, the control is shifted to the ERASE Label.
- In the erase label, the index of the pixel on the screen to be billed is stored in a var called idx. Now an inner loop –LOOP is created to fill all the pixels. The index value is loaded into the A register. Now M[A] is accessed and the value is changed to –1 which denotes black. This continues until all the pixels are filled. D stores the value of the number of pixels to be filled. (D starts from 8192). If it reaches 0, the loop terminates. Also, it is checked whether a key is held down during the loop execution. If yes then the loop continues. If not, then the control is handed over to the Erase part of the main loop.
- Similarly, the ERASE part is also implemented.

Screenshots:

(No key pressed)



(Key held down)



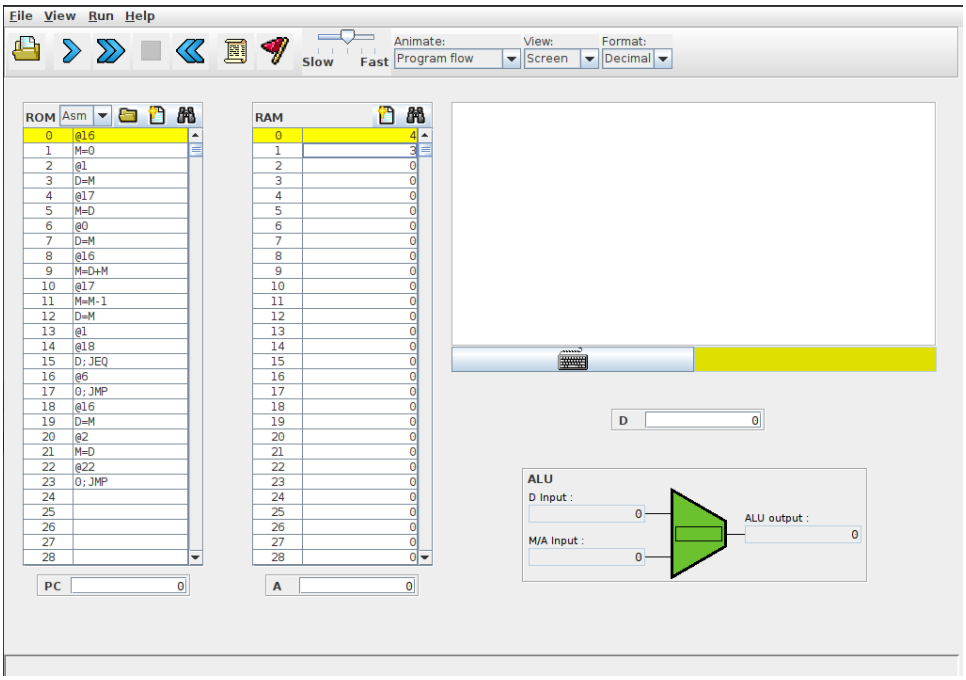
Mult.asm

Algorithm:

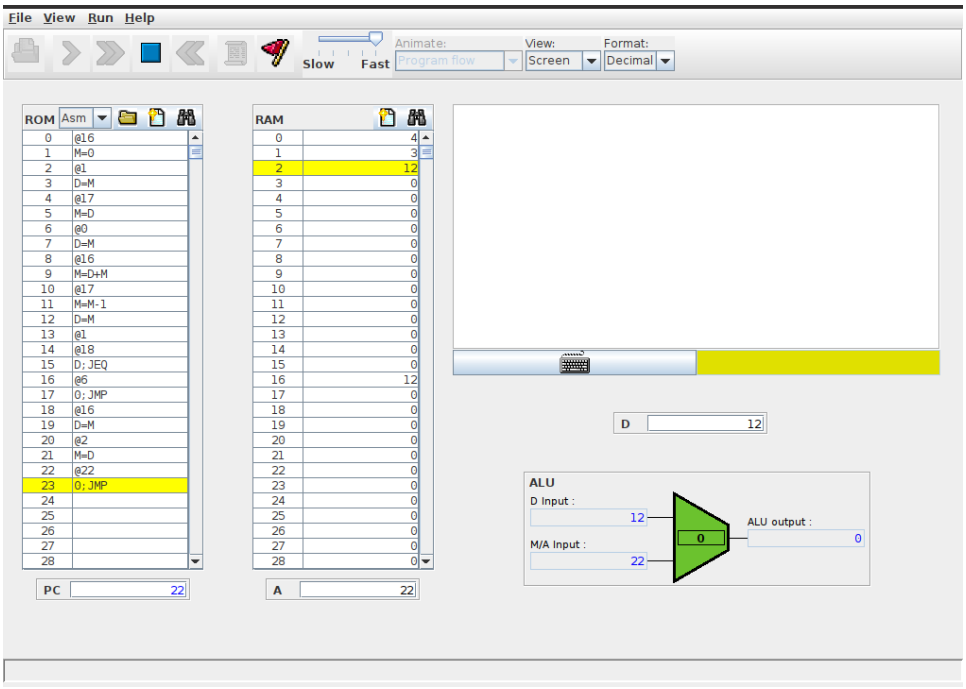
- The logic is implemented by using the concept of iterative additions. Two variables are defined stores the sum from multiple additions and i acts as the loop counter.
- A loop is started. The value of the first number is retrieved from the first value at index 0. Then the number Is added to the sum variable. Then the loop upper limit is decremented. When the loop counter reaches 0, the loop is terminated and the product is stored in index 2 of memory.

Screenshots:

(Before Multiplication)



(After Multiplication) - look at index 2 in RAM



PROJECT 5

Implementation of logic based on the slides from the nand2tetris course

Memory.hdl

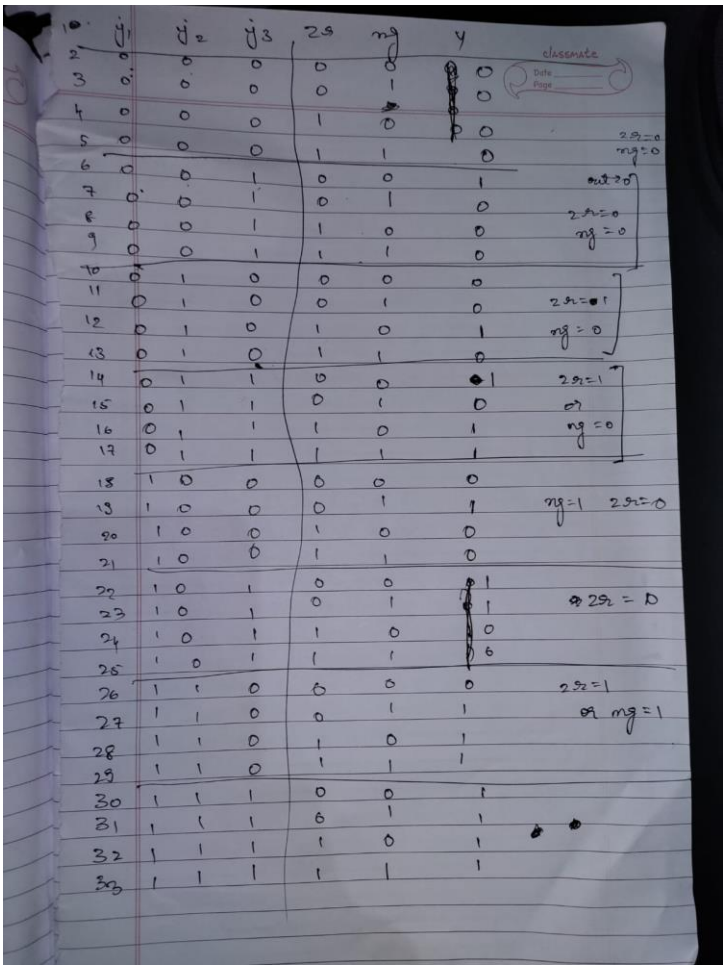
- A 4-way DMux is required in the beginning to select the path, Keyboard, RAM, OR Screen.
- Keyboard , RAM, OR Screen are enabled based on the output of the DMux
- Finally, the 4-way Mux is required to select among the outputs of Keyboard, RAM, OR Screen.
- The selector for the DMux and the Mux are the 13 and 14th bit of the address
- Tested using the Hardware Simulator and the test code

CPU.hdl

- The CPU requires ALU, MUXs, ARegister, Dregister, PC and logic gates to implement
- Mux is used to distinguish between A and C type instruction. The selector is the 15 th bit of the instruction
- All the internal connections are made based on the schematic diagram provided in the slides
- Jump is enabled only in certain cases. So, a combinational logic is created. The logic is derived from k-maps
- Tested using the Hardware Simulator and the test code
-

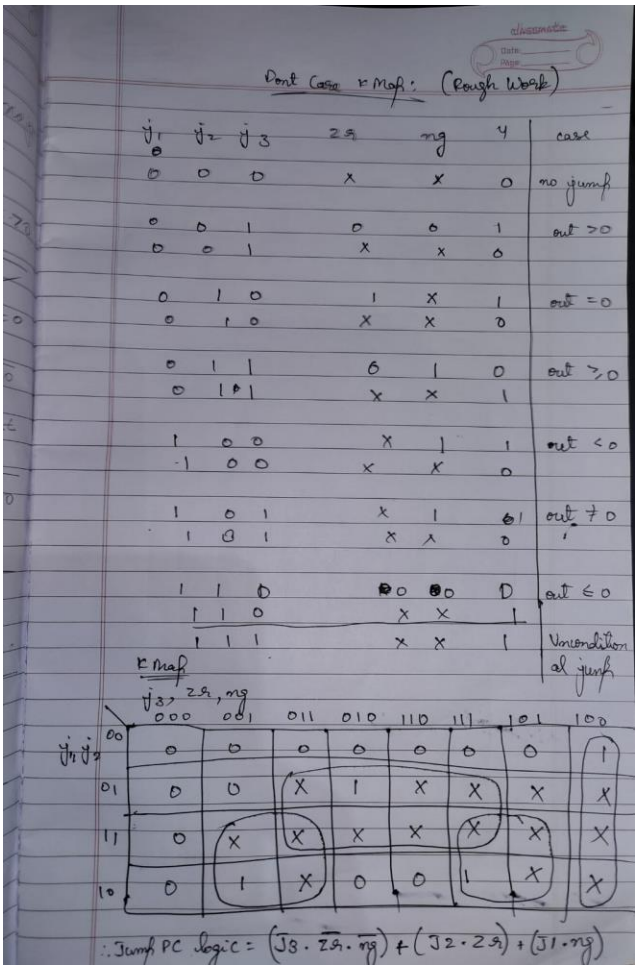
ROUGH WORK for JUMP LOGIC

(Without do not cares)



	y_1	y_2	y_3	z_2	m_2	y	
2	0	0	0	0	0	0	
3	0	0	0	0	1	0	
4	0	0	0	1	0	0	
5	0	0	0	1	1	0	$z_2=0$ $m_2=0$
6	0	0	1	0	0	1	
7	0	0	1	0	1	0	
8	0	0	1	1	0	0	$z_2=0$ $m_2=0$
9	0	0	1	1	1	0	
10	0	1	0	0	0	0	
11	0	1	0	0	1	0	$z_2=1$
12	0	1	0	1	0	1	$m_2=0$
13	0	1	0	1	1	0	
14	0	1	1	0	0	1	$z_2=1$
15	0	1	1	0	1	0	or
16	0	1	1	1	0	1	$m_2=0$
17	0	1	1	1	1	1	
18	1	0	0	0	0	0	
19	1	0	0	0	1	1	$m_2=1$ $z_2=0$
20	1	0	0	1	0	0	
21	1	0	0	1	1	0	
22	1	0	1	0	0	1	
23	1	0	1	0	1	1	$z_2=0$
24	1	0	1	1	0	0	
25	1	0	1	1	1	0	
26	1	1	0	0	0	0	$z_2=1$
27	1	1	0	0	1	1	or $m_2=1$
28	1	1	0	1	0	1	
29	1	1	0	1	1	1	
30	1	1	1	0	0	1	
31	1	1	1	0	1	1	
32	1	1	1	1	0	1	
33	1	1	1	1	1	1	

(With do not cares to minimize hardware)



Computer.hdl

- The Hack Computer requires the CPU, Memory and the ROM32K parts to function.
- All the internal nodes are internally establishing a logical connection between CPU, Memory, and the ROM.
- Tested using the Hardware Simulator and the test code

PROJECT 6

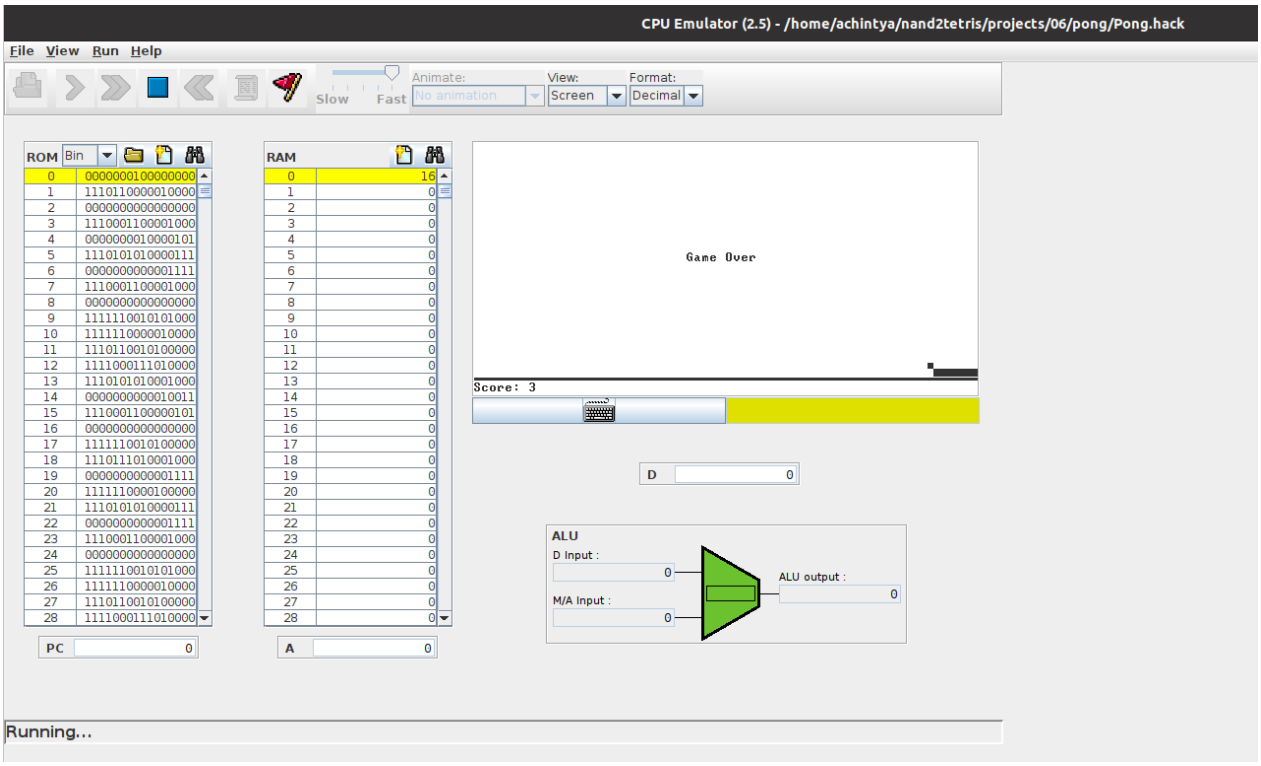
- Assembler for HACK ISA implemented using python.

Algorithm:

- The file is opened and the lines are read. The comments are removed by looking for '//' string line by line. Any string after // in the line is deleted. Next the white spaces are also removed from the line. All of these are implemented using python functions.
- Now dictionaries for opcodes are defined. They are symTab, dest, comp and jump. All the stores have the predefined opcodes of the HACK ISA. Also, another dictionary 'labels' is created. This stores the line number to which the control should shift after a label is called.
- Now the code, devoid of comments and white spaces is stored in a list, lines.
- The list lines are traversed and the labels are indexed with the line number the control should jump to when called. Also, the labels are removed from the program and the other instructions and stored in a new array new. On the new array is traversed and all the label calls are converted into integers referring to the symTabs and labels dics.
- Finally all the instructions are traversed. If it starts with '@' then it is A type. Then the number after is converted into a 15-bit binary and 0 is added in the beginning. if instruction has = then it is an arithmetic instruction the string before = gives the destination the string after gives the computation part now using the dest and the comp dictionaries the codes for the destination and the comp part is obtained if they are not present then the line is invalid according to Hack ISA Final string is obtained by concatenating the following in order. "111" as it is a C type instruction bits and bits 14 and 13 are always 1 bC the 7 6-bit computation part
- if ; is present then the instruction is jump type. The part before ; gives the computation part. The part after; gives the jump condition. The final binary string is calculated.
- The binary instructions are written into the respective .hack file.

(Screenshots and GitHub link in next page)

(Pong game using hack file)



GITHUB LINK:

https://github.com/achintyapro03/nand2tetris